Government College of Engineering, Jalgaon

(An Autonomous Institute of Govt. of Maharashtra)
Department of Computer Engineering

Student Name :_____ **PRN:** _____

Course Teacher **:** Mrs Priyanka H. Gadade, Government College of Engg., Jalgaon

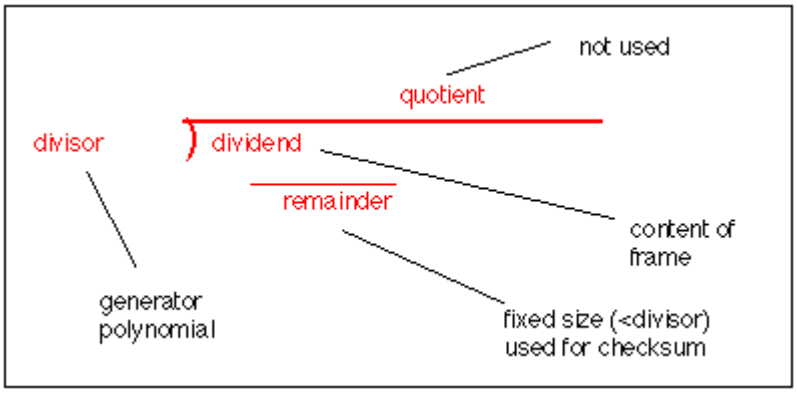
---

Experiment No. ____

Objectective: Implementation of Cyclic Redundancy Check using modulo-2 division method.

Theory:
A crc is a form of integrity checksum. It is a powerful method for detecting errors in the received data is by grouping the bytes of data into a block and calculating a Cyclic Redundancy Check (CRC). This is usually done by the data link protocol and calculated CRC is appended to the end of the data link layer frame.

The CRC is calculated by performing a modulo 2 division of the data by a generator polynomial and recording the remainder after division.



Cyclic Redundancy Check
0 + 0 = 0
0 + 1 = 1 + 0 = 1
1 + 1 = 0 (because 2 has remainder 0 after dividing by 2)
The multiplication table is equally simple:
*0 * 0 = 0*
*0 * 1 = 1 * 0 = 0*
*1 * 1 = 1*

What's more, subtraction is also well defined (in fact the subtraction table is identical to the addition table) and so is division (except for division by zero). What is nice, from the point of view of computer programming, is that both addition and subtraction modulo 2 are equivalent to bitwise *exclusive or* (XOR).

Cyclic Redundancy Check and Modulo-2 Division:

- ❑ CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in communication channel.
- ❑ CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

n : Number of bits in data to be sent from sender side.
k : Number of bits in the key obtained from generator polynomial.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use *modulo-2 binary division* to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same.

Receiver Side (Check if there are errors introduced in transmission): Perform modulo-2 division again and if remainder is 0, then there are no errors.

Modulo 2 Division:

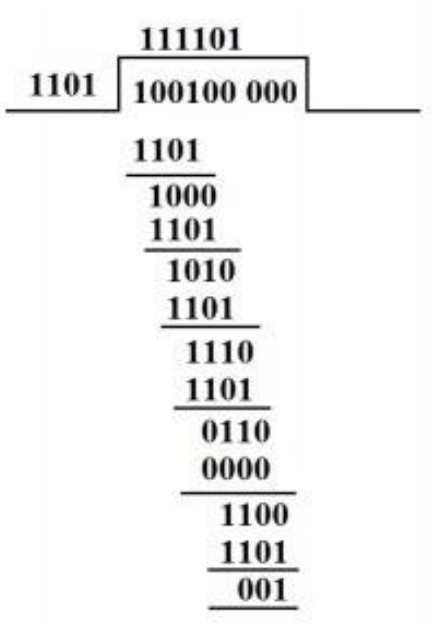
The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Example 1 (No error in transmission):

Data word to be sent - 100100
Key - 1101 [Or generator polynomial $x^3 + x + 1$]

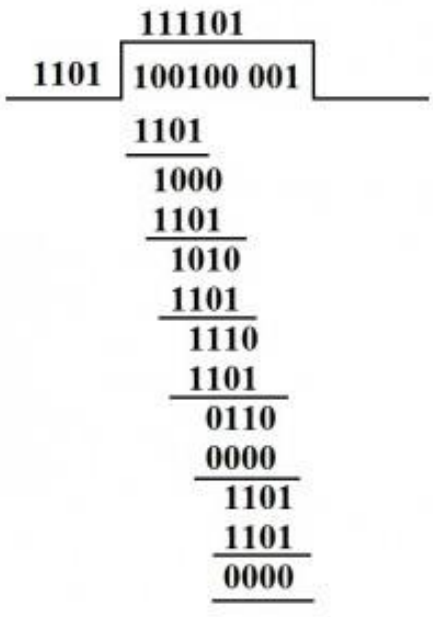
Sender Side:



Therefore, the remainder is 001 and hence the encoded data sent is 100100001.

Receiver Side:

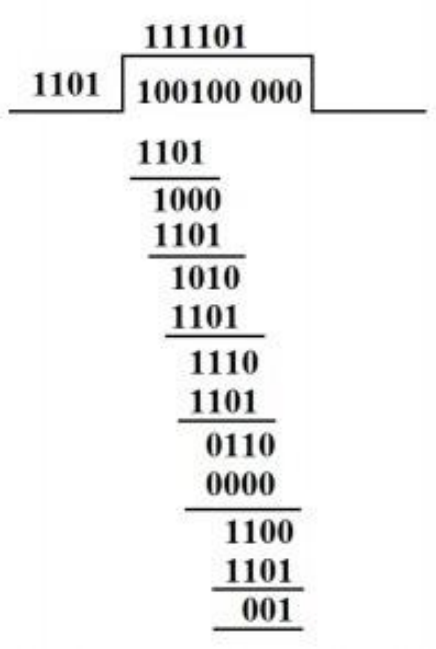
Code word received at the receiver side 100100001



Therefore, the remainder is all zeros. Hence, the data received has no error.

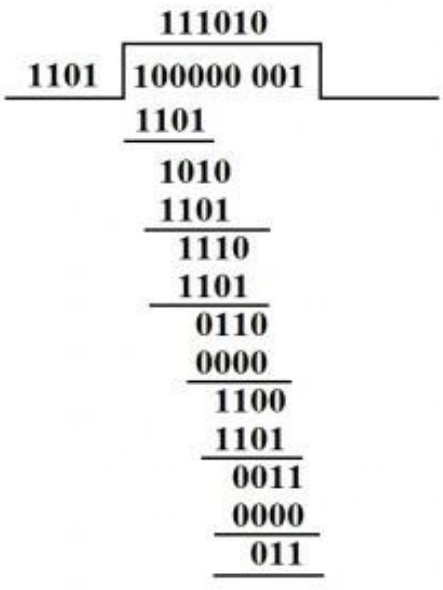
Example 2: (Error in transmission)
Data word to be sent - 100100
Key - 1101

Sender Side:



Therefore, the remainder is 001 and hence the code word sent is 100100001.

Receiver Side:
Let there be error in transmission media
Code word received at the receiver side - 100000001



Since the remainder is not all zeroes, the error is detected at the receiver side.

Implementation:

```
//Program for CRC
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int i, j,k = 0;
        int flag = 1, a[16], g[16], r[20],div[16], n, m;
        system("clear");
        printf("No. of Bite in CRC Generator : ");
        scanf("%d", &n);
        printf("\nEnter the CRC Generator : \n");
        for(i= 0; i<=n; i++)
        scanf("%d", &g[i]);
        printf("\nNo. of Bits in Data : ");
        scanf("%d", &m);
        printf("Enter Data : \n");
        for(i = 0; i<=m; i++)
        scanf("%d", &a[i]);

        if(m<n || (g[0] && g[n]) == 0)
        {
                printf("Not a proper generator \n");
                exit(0);
```

```
}

for(i = m+1; i<=m+n; i++)
        a[i] = 0;

for(j= 0; j<=n; j++)
        r[j] = a[j];

for(i= n; i<=m+n;i++)
{
        if(i>n)
        {
                for(j = 0; j<n; j++)
                r[j] = r[j+1];
                r[j] = a[i];
        }
        if(r[0])
                div[k++] = 1;
        else
        {       div[k++] = 0;
                continue;
        }
        for(j= 0; j<=n; j++)
                r[j] = r[j] ^ g[j] ;
}

printf("\nQuotient is : ");
for(j= 0; j<k; j++)
        printf("%d ", div[j]);

printf("\nReminder is : ");
for(i= 1; i<=n; i++)
        printf("%d ", r[i]);

printf("\nTransmitted frame  is : ");
for(i = m+1,j= 1; i<=m+n;i++, j++)
        a[i] = r[j];

for(i= 0; i<=m+n; i++)
        printf("%d ", a[i]);
```

```
printf("\n");

printf("\nNo. of bits of Data at Sender : ");
scanf("%d", &m);
printf("Enter Data at Sender : \n");
for(i = 0; i<=m; i++)
        scanf("%d", &a[i]);

for(j = 0; j<=n; j++)
        r[j] = a[j];
k = 0;
for(i= n; i<=m;i++)
{       if(i>n)
        {       for(j = 0; j<n; j++)
                        r[j] = r[j+1];
                r[j] = a[i];
        }
        if(r[0])
                div[k++] = 1;
        else
        {
                div[k++] = 0;
                continue;
        }
        for(j= 0; j<=n; j++)
                r[j] = r[j] ^ g[j] ;
}

printf("\nQuotient is : ");
for(j= 0; j<k; j++)
        printf("%d ", div[j]);

printf("\nReminder is : ");
for(i= 1; i<=n; i++)
        printf("%d ", r[i]);

for(i= 1; i<=n; i++)
{
        if(r[i])
        flag = 0;
```

```
        }

        if(flag)
                printf("\n No Error\n");
        else
                printf("\nError");

        return 0;
}
```

Output:

```
harish@harish-Inspiron-3537:~/harish$ ./a.out
No. of Bite in CRC Generator : 3
Enter the CRC Generator :
1
1
0
1


No. of Bits in Data : 5
Enter Data :
1
0
0
1
0
0

Quotient is : 1 1 1 1 0 1
Reminder is : 0 0 1
Transmitted frame  is : 1 0 0 1 0 0 0 0 1
```

```
No. of bits of Data at Sender : 8
Enter Data at Sender :
1
0
0
1
0
0
0
0
1

Quotient is : 1 1 1 1 0 1
Reminder is : 0 0 0
No Error
harish@harish-Inspiron-3537:~/harish$
```

Mrs. Priyanka H. Gadade
Course Teacher